Project HomeFinder

1st Ilian Sebti
Department of Engineering
Hanyang University
Seoul, Korea
Group HomeFinder
ilian.sebti@edu.devinci.fr

2nd William Olsson
Department of Engineering
Hanyang University
Seoul, Korea
Group HomeFinder
lobbeolsson@gmail.com

3th Jeong JinYeong
Department of Engineering
Hanyang University
Seoul, Korea
Group HomeFinder
tttjjjyyy1234@daum.net

4th Enkhjin Puntsagnorov
Department of Information systems
Hanyang University
Seoul, Korea
Group HomeFinder
enhjinnasaa48@gmail.com

5rd Nina Lauria
Department of Information systems
Hanyang University
Seoul, Korea
Group HomeFinder
ninalauria01@gmail.com

***Abstract*—For our project, we would like to develop a new website that shifts the focus of property search away from geographical features and onto the immediate surroundings of a potential new home. The purpose of this project is to refine and tailor the apartment search experience to the users' needs. Users should be able to filter their search based on self-selected criteria to obtain the best results for them. The website should have the capability to find all available listings that meet the user's chosen filters. The results will be displayed on a map, and users can save these results to their favorites for future reference. The advantage of such a search is that users won't have to conduct their own research to discover what's nearby the apartment. With the map-based search, it's easy for users to see the Points of Interest in the vicinity of the location. With enough input data, the website can suggest the best home for the user, making the apartment search more enjoyable and time-saving.**

I. Role Assignments

Roles	Name	Task description and etc.
User	Nina Lauria	Users navigate through the platform, employing various filters to refine their home searches, while also utilizing functionalities like sorting and favoriting to enhance their experience. They may create accounts for personalized interactions and seek assistance through help sections when needed.
Customer	Ilian Sebti	Customers, such as real estate agencies or sellers, manage their property listings on the platform, ensuring accurate and appealing presentation. They utilize provided analytics and insights to understand market trends and user behavior, adjusting their strategies to enhance visibility and engagement accordingly.
Software developer	Jeong JinYeong / Enkhjin Puntsagnorov	Software developers engage in coding, testing, and optimizing the platform, ensuring its functionalities from search features to data security are robust and user-friendly. They collaborate with stakeholders to align technological implementations with the platforms vision, ensuring scalability and adaptability to evolving technologies.
Development manager	William Olsson	Development managers oversee the platform's development process, ensuring alignment with quality and requirements. They manage task distribution among the software development team, facilitate resolution of challenges, and ensure the project adheres to timelines and budgets, while maintaining a focus on quality and user experience.

II. Introduction

A. Motivation

Our motivation initially sprung from our personal struggles and those of others around us. The collective frustration regarding the conventional online house hunting experience, the endless tabs, the disjointed correlation between home and locality, and the exhaustive manual research struck a chord with us. We realized that while physical properties of a home (such as size, number of rooms, etc.) are pivotal, the impact of location features on the living experience was a significantly underrepresented factor in existing platforms.

We believe that a home is not just a physical space but an environment that significantly influences one's quality of life. Recognizing the paramount importance of convenience,

accessibility, and connectivity in modern living, we were driven to devise a solution that prioritizes these aspects. Our platform is not just a tool; it's a companion that assists users in finding a home that resonates with their lifestyle, needs, and preferences.

We were also motivated by the prospect of empowering users to make well-informed decisions. By presenting a visual, map-based experience, we offer users a comprehensive view that enables them to evaluate and compare homes based not just on the properties but also on the surrounding infrastructures, such as transportation links, grocery stores, schools, and hospitals. This integrative approach ensures that users can visualize their life in a potential home, foreseeing the conveniences and challenges that the location presents.

B. Problem Statement

The process for creating the problem statement involved several steps. Initially, we identified the problem area of apartment hunting, focusing on the issue that location features are often not adequately considered when searching for a new home. Next, we analyzed the causes and effects of the problem and developed a potential solution.

Following that, we conducted a user forecast to determine our primary target audience and identify other potential beneficiaries of our product. The user forecast also aimed to validate whether there is indeed a demand for such a product.

Online searching for a new abode generally bases itself on the physical properties of the house or apartment. Prioritizing these properties over location features results in an incomplete picture and search experience. Even though the search results are displayed on a map, you still have to research manually to find out where the necessary things are located near your potential new home.

For this reason, we create a platform that turns the search for a new place to live into a visual, map-based experience. It is easy to see which Points of Interest are located near the places you are looking at, and you also see how to get there. Moreover, with sufficiently input data, we could actually suggest the best home for the user.

The solution is suitable for users who do not have to live in a specific place but rely on the availability and connection (transportation) around the property. Companies can use the aggregated data from the real estate platform to understand needs and incorporate the insights into real estate development and real estate pricing.

III. Related Software

A. Zillow

Zillow is an online real estate marketplace that provides a plethora of data related to available properties, including price estimates, aerial views, and comparative market analyses.

While Zillow does present a wide array of property options with basic location mapping, our platform aims to go a step further by intensively integrating location features into the search experience. Zillow tends to prioritize physical property features and pricing data, whereas our platform is envisaged to offer a visual, map-based searching experience that prominently highlights Points of Interest (POIs) and demonstrates how to navigate to them.

B. Zigbang

Zigbang stands as the premier real estate Service in South Korea, boasting over 30 million users and a wealth of user data. Not only does it sustain a dominant market presence, but it is also dynamically expanding its offerings by introducing an array of services such as 'IoT service' and 'VR Home Tour service', complementing their staple 'on-tact real estate sales service'. Zigbang notably runs a 'real estate recommendation service' which closely mirrors our 'HomeFinder', characterized by:

- Recommending properties using parameters like school districts, transportation accessibility, and selling prices.
- Enabling users to stay abreast of new listings in real-time via the Favorites feature.

Despite its strengths, Zigbang's recommendations hinge on a limited set of criteria, such as school district and selling price. In contrast, HomeFinder enhances the personalization of its service, recommending properties based on an expanded range of criteria that also encompasses nearby gyms, cafes, and convenience stores.

C. Dabang

Dabang, the second-leading real estate Service in South Korea, has secured 20 million users, providing it with a substantial data reservoir, albeit not as extensive as Zigbang's. Dabang excels in customization within its fundamental service, "on-tact real estate sales service", standing distinctively in comparison with Zigbang by recommending properties based on a multifaceted set of criteria: selling price, property type (rental, lease, sales), square footage, number of floors, parking availability, and elevator availability, among others.

Nonetheless, the service is not without its shortcomings, chiefly that its criteria predominantly spotlight the property's internal characteristics. While the internal facets of a residence are undeniably significant, external factors like proximity to convenience stores and gyms also wield importance in enhancing life quality. Hence, HomeFinder aims to uplift user satisfaction by recommending properties that judiciously consider both internal and external conditions.

D. Naver Real Estate

Naver Real Estate, being the third largest real estate service in South Korea, carries its own set of strengths concerning capital, brand recognition, and a substantial user pool, owing to its inception by Naver, a prominent IT conglomerate. Unlike its counterparts, this service prioritizes information dissemination over direct transactions. Leveraging the robust platform of Naver, it dominates in real estate listings, claiming a remarkable 95 percent share.

Nevertheless, like Zigbang and Dabang, Naver Real Estate concentrates solely on providing a filtering function for the internal conditions of a property and makes uploading photos to

visualize property conditions a challenging task. HomeFinder, therefore, aspires to boost user satisfaction by recommending properties with a balanced consideration of both internal and external conditions, and facilitating the provision of photos to assess the property's condition.

IV. Requirements

The information architecture of the website is well-structured and logically constructed. The main categories of the website are the homepage, the resultpage, the detail page, and the contact page.

A. Homepage

- The website must be capable of filtering user searches based on selected criteria, such as location, price range, property type, number of rooms, etc.
- Users should be able to select as many search filters as they wish to refine their search results according to their preferences and needs.
- Intuitive and interactive user interface components should be employed to facilitate easy navigation and usage of search filters.

B. Result Page

- Results should be displayed along with a dynamic map to provide users with a clear spatial understanding of the property locations relative to various Points of Interest.
- The map should be interactive, allowing users to zoom, pan, and click on property markers for more detailed information.
- Users should have the ability to sort search results based on various parameters, such as price (ascending and descending), proximity to Points of Interest, property size, etc.
- The sorting functionality should be straightforward, enabling users to quickly rearrange their search results according to the selected parameter.

C. Favorites Page

- Users should be able to add listings to their favorites for easy access and future reference.
- The favorites page should provide a simplified view of the selected properties, with options to visit the detailed view or remove items from the favorites list.
- It would be beneficial to allow users to categorize or label their favorites for enhanced organization and retrieval.

D. Account Creation Page

- Users should be able to create an account using a simple and secure sign-up process, with necessary validations to ensure data accuracy and security.
- Account management options should be provided to allow users to update their personal information, preferences, and password.

E. View of Nearby Services

- "Nearby Services" provides users with detailed information about essential amenities, such as grocery stores, schools, healthcare facilities, and public transportation options in the vicinity of rental properties.
- Users can access comprehensive details about these services, including their operating hours.

F. Rental term

- User can filter the duration of rent. So that available apartment or house will be shown as time by time. This filter allowing users to select the duration of the lease they are looking for.

G. Accessibility

- User should be able to see the accessibility of apartment.
- Accessibility filter provide a quick view of stairs and elevator information. For each apartment list, collect and store information about the number of floors and the presence or absence of elevators in that apartment.

V. Development environment

A. Choice of software development platform

In orchestrating our development environment, we have strategically selected each technology to optimize our workflow for peak performance, flexibility, and efficiency. **For the frontend, React.js emerges as our choice** for its state-of-the-art capabilities in crafting dynamic user interfaces that engage and respond with agility. Our commitment to a seamless user experience is further evidenced by the integration of **HTML and CSS**, which serve as the bedrock for a solid and aesthetically pleasing design framework.

The backend architecture is entrusted to the robust Node.js, coupled with Express.js for its lightweight and flexible framework that accelerates our backend development. We strategically incorporate **Java** for certain backend modules, capitalizing on its storied reliability and inherent security features, which are paramount for a cross-platform enterprise application.

Data management is the backbone of any modern web application, and MongoDB stands as our database of choice. Its agile NoSQL structure allows us to adeptly handle and scale with the large, unstructured datasets that our application generates and consumes.

Development tools are the artisans' instruments, and **Visual Studio Code** is our chosen palette, offering unmatched support for a wide array of programming languages and extensions that empower our developers to write, debug, and deploy with confidence. Essential to our process are tools like **TODO Tree and CodeSee,** which bring clarity and organization to our codebase, and **Insomnia**, which serves as our gateway for thorough API testing and debugging. The collaborative nature of our project is facilitated by **GitHub**, a cornerstone for our version control systems, ensuring that our code evolves in harmony with multiple contributors.

A robust development environment is more than software; it's also the hardware that supports it. Hence, we mandate a minimum of 8 GB of RAM for our systems to ensure that our suite of development tools operates at full stride without bottlenecks, allowing our team to maintain a smooth and efficient development cadence.

B. Cost estimated

Given the nature of our project, the focus is on cost-efficiency; therefore, we aim to utilize free resources wherever possible. This commitment extends to our development environment, where we are using Visual Studio Code—an open-source code editor—across all operating systems.

C. Software in Use

Our development toolkit comprises specific, up-to-date versions of software essential for our project's success:

- Employing React.js version 17 or later, we ensure that our front-end architecture is both contemporary and performant.
- With Node.js version 16 or above, our back-end is scalable and responsive to real-time data flow.
- Express.js complements Node.js, offering a streamlined framework for our web application's infrastructure.
- MongoDB, a NoSQL database, is our choice for efficient data storage and retrieval, pivotal for handling our application's data demands.
- Visual Studio Code serves as the cornerstone of our development, with its extensibility and diverse language support, including vital extensions for productivity.
- Insomnia is integral to our API lifecycle, ensuring that each endpoint is robustly tested and debugged.
- GitHub underpins our collaborative efforts, providing a robust platform for code versioning and team collaboration.

The operating systems supporting our development include Windows 11, macOS, and Linux. We require a minimum of 8 GB of RAM for our computer systems to ensure efficient multitasking and smooth operation of our development tools.

D. Task distribution

Name	Roles	Task description
Nina Lauria	Documentation and Frontend Leader	Frontend tasks and responsible for UI testing and feedback. Also oversees the documentation to ensure it's updated and does frontend tasks.
Jeong JinYeong / Ilian Sebti	Backend DevOps	Backend tasks like setting up the server, database integration, API design and implementation.
William Olsson	Project and Backend leader	Integration of Frontend and Backend, does also DevOps tasks in both.
Enkhjin Puntsagnorov	Frontend DevOps	Is tasked with the development and optimization of the frontend, ensuring the user interface is both functional and user-friendly.

VI. Specifications

The required functionalities of our project are divided between multiple web pages.

A. Homepage / Search page

Through this page, users will be able to search for a home by using a filter based search engine. The selectable criterias for the filters will be mostly property oriented (City, Price, number of rooms, energy efficiency...). Users should be able to select multiple filters to refine their search results. We did clickable Mockups for better visualization. To see the Mockups click here.

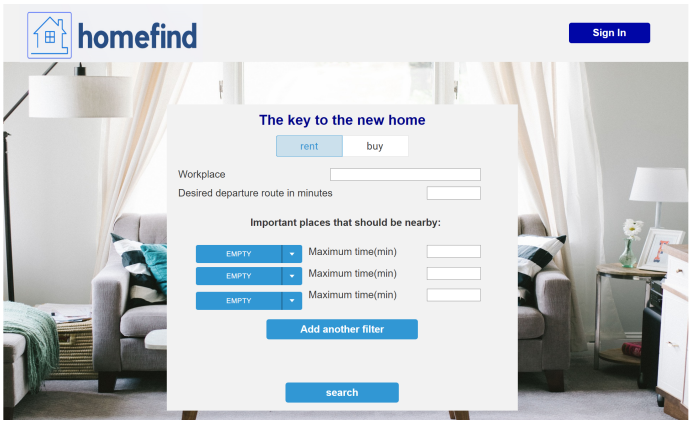

Fig. 1: Prototype of the Homepage / Search page

Backend Implementation:

- Define API endpoints for property search.
- Implement filter-based search logic, which includes receiving filter parameters and querying the database for matching properties.
- Return the search results in JSON format to the frontend.

Frontend Implementation:

- Create the search form with filter options (City, Price, Number of Rooms, etc.).
- Capture user input and make an API request to the backend.
- Display the search results on the frontend, possibly as a list of properties that meet the criteria.

B. Result Page

This page will display the results of the search on a dynamic map, giving the user the ability to rearrange the search results and give a clear spatial understanding of the property locations relative to various Points of Interest. From this page, users will be able to add properties to their favorites. Backend Implementation:

- Create an API endpoint to fetch the list of properties for the result page.
- Implement logic to retrieve property details, including geolocation information.
- Return the property details in JSON format to the frontend.

Frontend Implementation:

- Display the search results on a dynamic map, using JavaScript mapping libraries like Mapbox or Google Maps.
- Allow users to rearrange search results on the map.
- Implement the option to add properties to the user's favorites list, sending requests to the backend to update the user's favorite properties.

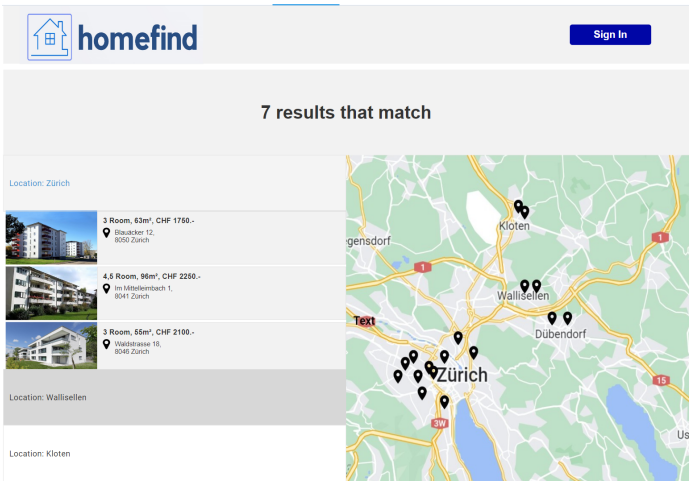

Fig. 2: Prototype of the Result Page

C. Favorites

From this page, users will be able to view a listing of their favorite properties as well as a simplified view of them with the option to access the detailed view or to remove the property from the list. Frontend Implementation:

- Create a user interface for viewing a listing of favorite properties.
- Implement options for users to access the detailed view of properties or remove them from the list.
- Send requests to the backend to manage the user's favorite properties.

D. Account creation/login

Backend Implementation:

- Implement user authentication and authorization mechanisms, using libraries like Flask-Login or Django's authentication system.
- Create API endpoints for user registration, login, and account management.

Frontend Implementation:

- Create user registration and login forms.
- Capture user input, make API requests to the backend for registration and login, and manage user sessions.
- Implement UI elements for user account management, including a user profile page.

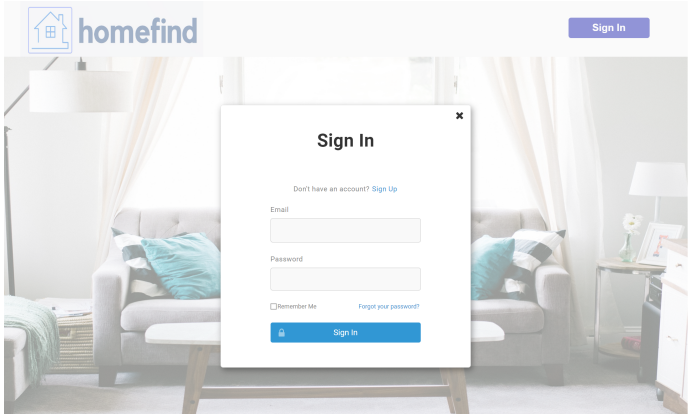

Fig. 3: Prototype of the Account creation/login

E. View of nearby Services

Use third-party APIs (e.g., Google Places API) to retrieve data on amenities like grocery stores, schools, healthcare facilities, and public transport and design database schema to store information of amenities linked to property locations.

Backend Implementation:

- Develop an endpoint that takes property location as input and queries the table for nearby services.

Frontend Implementation:

- Implement a map interface (using libraries like Leaflet.js or Google Maps JavaScript API) that displays nearby services as markers.
- Create a UI component that shows a detailed list of amenities when a user selects a property.

F. Rental Term

The rental term feature requires a UI component that allows users to define the duration of the lease they are interested in. This could take the form of a slider or a set of dropdown options. The property search functionality would be enhanced to interpret the rental term as a search parameter, filtering the database results accordingly.

Our database schema would be structured to include a 'rentalTerm' field within the properties table, which would reflect the available durations of lease for each listing. Automated tests would be crucial to ensure the filter operates as intended, only showing users the properties available for their specified term.

G. Accessibility

Accessibility information is pivotal for many users. To accommodate this, property input forms would be updated to capture details regarding elevators and the number of floors. The database schema would evolve to include an 'accessibilityFeatures' table, which would store this information linked to each property.

We would develop backend logic to provide this information through an API endpoint, which would be called when a user views property details. The frontend would display this information clearly, using universally recognized icons or labels to indicate accessibility features.

VII. Architecture Design and Implementation

A. Overall architecture

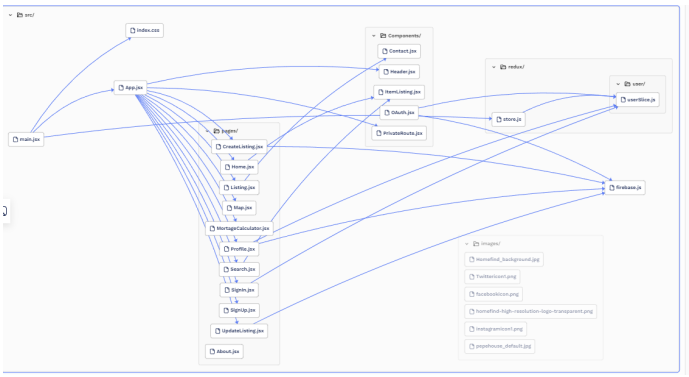

Fig. 4: architecture Frontend

In the frontend diagram, the structure of a React application is displayed, with 'App.jsx' at the core, bringing together various pages and components. The pages are organized as distinct units such as 'Home.jsx', 'Listing.jsx', and 'Mortgage-Calculator.jsx' under 'pages/', indicating one route per page. Components like 'Header.jsx' and 'OAuth.jsx' are reusable and utilized across the application to ensure a consistent user interface. The Redux logic, evident in 'store.js' and 'userSlice.js' under 'redux/', manages the global state and facilitates efficient data handling and UI updates. The presence of 'firebase.js' suggests the use of Firebase services, potentially for authentication or database interactions.

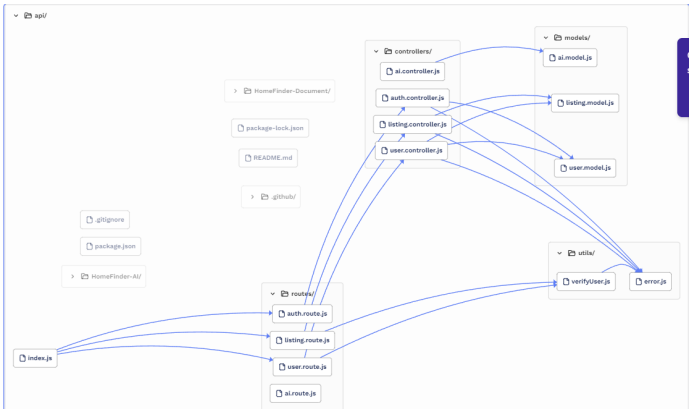

Fig. 5: architecture Backend

The backend architecture diagram reveals a Node.js application structure that delineates models, routes, and controllers, ensuring a clean separation of concerns. It illustrates how various routes such as 'auth.route.js', 'listing.route.js', 'user.route.js', and 'ai.route.js' are linked to their respective controllers. These controllers, in turn, communicate with their specific models – 'ai.model.js', 'listing.model.js', and 'user.model.js' – which define the data structure for the application. Auxiliary services like 'verifyUser.js' and 'error.js' are also incorporated, suggesting middleware used for authentication tasks and error handling. The central 'index.js' acts as the entry point, orchestrating the interplay of all these components.

VIII. Directory organization - Frontend

To develop Homefinder, we used visual studio so that a it can be used in web-based environment. Frontend serves as a role in providing interface to a user. It connects user and server by providing input fields that a user can enter information. Once this job is done, the values are delivered to backend. Also, Frontend brings data in the database and provides it to the user after changing it to a data that a user can understand.

Location of code is here.

Directory	File Name	Library
HomeFinder-Frontend	images src .eslintrc.cjs README.md index.html package-lock.json react-router-dom package.json postcss.config.js tailwind.config.js vite.config.js	react @reduxjs/toolkit firebase react @tailwindcss/line-clamp @types/react @types/react-dom @vitejs/plugin-react-swc autoprefixer eslint eslint-plugin-react eslint-plugin-react-hooks eslint-plugin-react-refresh postcss tailwindcss vite
HomeFinder-Frontend/src	App.jsx firebase.jsx index.css main.jsx	react-router-dom firebase/app redux-persist/integration/react
HomeFinder-Frontend/src/Components	Header.jsx ItemListing.jsx OAuth.jsx PrivateRoute.jsx	react react-router-dom
HomeFinder-Frontend/src/pages	About.jsx CreateListing.jsx Home.jsx Listing.jsx Map.jsx MortgageCalculator.jsx Search.jsx SignIn.jsx SignUp.jsx UpdateListing.jsx	react swiper/react swiper/modules react-icons/io react-router-dom react-redux
HomeFinder-Frontend/src/redux/user	userSlice.js	@reduxjs/toolkit
HomeFinder-Frontend/src/redux	store.js	@reduxjs/toolki redux-persist
HomeFinder-Frontend/images	Homefind_background.jpg Twittericon1.png facebookicon.png homefind_high-resolution_logo_transparent.png pepehouse_default.jpg instagramicon1.png	

Fig. 6: Directory Organization - Frontend

Class component:

- **images:** In the images folder, there are stored images, fonts, logo of page and other resource files that used in homefinder web.

- **src folder:** in this folder, displays the main navigation bar/menu, including links to different sections of the website for example the home, listings and contact information of the use.

A. src/pages/About.jsx

The code represents an ”About” component that displays information about Homefind. It contains information about the Homefind web service, highlighting its unique approach to property search.

Libraries Used and Their Purpose:

- import React from ”react”: it provides the core functionality for building the user interface in React applications.
- import useState from ”react”: it enables state management within functional components in React, allowing the component to manage and update its state.
- import getDownloadURL, getStorage, ref, uploadBytesResumable : Purpose for used that facilitates interactions with Firebase Storage, allowing the uploading and retrieval of files (in this case, images).
- import app from ”../firebase”: Imports the Firebase application instance where Firebase services are initialized (used for Firebase Storage operations in this code).
- import useSelector from ”react-redux”: Connects React components to the Redux store, allowing access to the application’s state managed by Redux.

Code Purpose: The code represents a form component (CreateListing) used to create a listing with various fields, image uploads, and submission logic:

- State Initialization: Utilizes useState to manage various states like files, formData (holding listing data), imageUploadError, uploading, error, and loading.
- Image Upload Handling: Allows users to upload images (maximum 6) for the listing, validating image counts, sizes, and managing image URLs in the form data.
- Firebase Storage Integration: Handles storing images to Firebase Storage using Firebase SDK’s storage functionalities (storeImage function).
- Form Field Handling: Manages form inputs, checkboxes (sale, rent, parking, furnished, offer), textual inputs (name, description, address), and numeric inputs (bedrooms, bathrooms, regularPrice, discountPrice) using the handleChange function.

B. src/pages/Home.jsx

Libraries Used and Their Purpose:

- import Link, useNavigate from ”react-router-dom”: Provides components for routing and navigation within a React application.
- import Swiper, SwiperSlide from ”swiper/react”: Allows the use of state and lifecycle features in functional components.
- import Navigation, FreeMode, Pagination from ”swiper/modules”: Offers a swiper/slider component for React applications, enabling the creation of slide-based interfaces.
- import IoIosArrowForward from ”react-icons/io”: Provides a collection of icons as React components.
- import Navigation, FreeMode, Pagination from ”swiper/modules”: Additional modules for Swiper to add functionalities like navigation, free mode, and pagination.
- import SwiperCore from ”swiper”: Core functionality for the Swiper package, enabling the usage of specific modules.

Code Purpose:

- The code represents the Home component, which serves as the landing page for a real estate application.
- Rendering Content: Displays a banner with a call-to-action to encourage users to start searching for properties. It created to presents the latest property listings in a Swiper component, allowing users to browse through property images and navigate to the respective listing details.
- Quick Links: Provides a section for quick links or navigation items, intended to direct users to important sections of the website or external resources.

C. src/pages/Listing.jsx

Libraries Used and Their Purpose:

- import FaBath, FaBed, FaChair, FaMapMarkerAlt, FaParking, FaShare, from ”react-icons/fa”: Provides specific icons from FontAwesome as React components.

Code Purpose:

- The code represents the Listing component responsible for displaying details of a specific property listing.
- Image Slider: Renders a Swiper component to display the images associated with the property listing. When clicking an image, it opens a modal to show an enlarged view to the user.
- Listing Details: it shows various details of the property listing, including its name, price, location, type, description, and amenities (bedrooms, bathrooms, parking, furnishings .
- The component essentially retrieves and displays detailed information about a property listing, including images, pricing, location, and various attributes, while also facilitating sharing and contact with the listing owner when applicable.

D. src/pages/MortageCalculator.jsx

Libraries Used and Their Purpose:

- React: Fundamental library for building user interfaces in React applications.

Code Purpose:

- The MortgageCalculator component is designed to calculate and display the estimated monthly mortgage payment based on user-provided input.
- Implements the ¡calculateMortgage¿ function to compute the estimated monthly mortgage payment based on the provided input values for loan amount, interest rate, and loan duration.

- State Management: Initializes state variables to store input values (homeValue, downPayment, loanAmount, interestRate, loanDuration, monthlyPayment) and a state for indicating when the calculation is in progress (calculating).
- UI Display: Renders a UI structure within a styled container to present the Mortgage Calculator title, input fields, and a calculated monthly payment display area.
- Calculation Execution: Initiates the mortgage calculation when the form is submitted, validates the input fields, performs the calculation, and displays the resulting monthly payment if it's successfully computed.

E. src/pages/Profile.jsx

Libraries Used and Their Purpose:

- React-Redux : Facilitates the integration of Redux state management with React components.
- Firebase Storage (Firebase SDK) : Handles file storage functionalities for uploading, downloading, and managing files in Firebase.
- React Router DOM: Manages routing and navigation in a React application, enabling navigation between different components based on URLs.

Code Purpose:

- State Management: Initializes state variables to manage user data, file upload, form data, and UI display flags (file, filepercentage, fileUploadError, formData, updateSuccess, showListingsError, showListings, userListings).
- File Upload: Manages file upload functionality using Firebase Storage, tracking upload progress and updating the form data with the uploaded file's URL.
- Form Handling: Handles form inputs for username, password, and email, allowing users to update their profile information. It also provides a button for users to update their profile details.
- User Actions: Includes functions for updating user information, deleting a user account, signing out, and handling the display of user listings.
- User Listing Display: Displays a user's listings if available, allowing users to view, delete, or edit their listings. The functionality also includes toggling the visibility of listings.
- UI Rendering: Renders a structured UI within a styled container to present the user profile information, profile image, form inputs, action buttons (delete account, sign out), and user listings with related functionalities.

F. src/pages/Search.jsx

Libraries Used and Their Purpose:

- React Hooks (useState, useEffect): Facilitates managing state and side effects in functional components.
- React Router DOM (useNavigate): Provides navigation functionality for routing in React applications.

Code Purpose:

- The Search components offers a comprehensive search interface allowing users to filter listings based on various criteria like type, amenities, and sorting options, and displays the results dynamically based on the selected filters. Additionally, it enables users to load more listings if available.
- It initializes state variables to manage various aspects of the search and display process
- Implements ¡onShowMoreClick¿ to fetch more listings when a user clicks the "Show more" button, extending the existing list.
- ¡handleChange¿ manages changes in form inputs for search term, type (rent, sale, all), checkboxes for amenities (parking, furnished, offer), and sorting options. It updates the ¡sidebardata¿ state accordingly.

G. src/pages/SignIn.jsx

Libraries Used and Their Purpose:

- React Hooks (useState): Manages state within functional components.
- React Router DOM (Link, useNavigate): Provides navigation functionality for routing in React applications.
- React Redux (useSelector, useDispatch): Facilitates interaction with the Redux store, allowing component-level access to state and dispatching actions.

Code Purpose:

- Sign in component serves as an interface for users to sign in by providing necessary credentials, interacting with the backend to authenticate users, and using Redux for managing loading states and error handling during the sign-in process.
- Form Handling provides form inputs for email and password, handling changes with the handleChange function, updating the ¡formData¿ state accordingly.

H. src/pages/SignUp.jsx

Libraries Used and Their Purpose:

- Used the same library as signin

Code Purpose:

- Sign up component serves as an interface for users to sign up by providing necessary credentials, interacting with the backend to register users, and displaying appropriate messages based on the registration outcome.
- Form Submission: Submits the form data to the server-side endpoint (/api/auth/signup) via a POST request. Upon receiving a response from the server:
- Form Handling function provides input fields for username, email, and password.
- If the registration is unsuccessful , it sets the error message received from the server.
- If the registration is successful, it navigates the user to the sign-in page.

I. src/pages/UpdateListing.jsx

Libraries Used and Their Purpose:

- React: A JavaScript library for building user interfaces.

- Firebase Storage: Utilized for handling file uploads and storage.
- React Redux: Used for managing application state, particularly user-related data.
- React Router DOM: Enables routing functionality in a React web application.

Code Purpose:

- Update Listing component combines React's state management, Firebase Storage for file handling, React Redux for user data, and React Router DOM for routing to facilitate the update of a property listing with various details and images.
- Form Submission handles form submission upon updating the listing. Validates input data and sends a POST request with updated listing information to the server. Upon success, navigates the user to the updated listing page. Handles errors and displays appropriate messages based on the submission outcome.
- Image Handling allows users to upload images for the listing. Images are uploaded to Firebase Storage Backend.

J. src/Components/Contact.jsx

Libraries Used and Their Purpose:

- import Link from "react-router-dom": This library provides a declarative way to create links between various pages or components, enabling users to interact with the application without causing a full page refresh. it updates the URL and renders the corresponding component without reloading the entire page.
- import useEffect, useState from "react : This library helps to managing side effects and local component state, respectively, facilitating dynamic updates and interactions within functional components. These hooks collectively enable components to handle asynchronous operations, perform actions on component mount/update, and manage state changes, enhancing the reactivity and functionality of the components.

Code Purpose:

- This code implements a contact feature allowing users to send messages to property landlords. It fetches landlord details based on the provided listing information, enables users to input messages, and generates a link using the landlord's email and the message content, facilitating direct email communication through a 'Send Message' button.

K. src/Components/Header.jsx

Libraries Used and Their Purpose:

- import FaSearch from "react-icons/fa": this library provides a set of pre-built icons for use within React components.
- import Link, useNavigate from "react-router-dom" : it facilitates navigation and routing within the React application.
- import useSelector from "react-redux": this library manages application state and enables access to the Redux store's state.

Code purpose:

- This code implements a header component for a home-finder web application. Constructs a header with a navigation bar containing links for Home, About, Map, Mortgage Calculator, and a Sign-in link or user avatar. For search functionality, it provides a search input field allowing users to search for properties. It captures the search term, updates it in the URL, and navigates to the search page with the entered query string when submitted.

L. src/Components/ItemListing.jsx

Libraries Used and Their Purpose:

- import Link from "react-router-dom" :
- import MdLocationOn from "react-icons/md": This allows for utilizing the Material Design icon for location within a React application, providing visual representation or indication related to location-based functionalities or information.

Code Purpose: This code defines a reusable component for displaying individual property listings such as Listing display, Link to details, dynamic rendering and conditional rendering.

- Listing display: it displays information about a property listing, including its image, name, address, description, pricing details, and bedroom/bathroom count.
- Link Listing details: Wraps the listing details in a ¡link¿ component, enabling users to click on the item to navigate to the specific details page of that listing.
- Visual Enhancement: Utilizes the MdLocation icon from react icons to display a location icon before the property address, enhancing the visual representation of the address information.
- Conditional Rendering: Conditionally displays additional information, like the offer price or type of listing (e.g., "per Month" for rentals), based on the listing's attributes and type.

M. src/Components/QAuth.jsx

Libraries Used and Their Purpose:

- import GoogleAuthProvider, getAuth, signInWithPopup from "@firebase/auth": This library provides authentication services, allowing users to sign in with Google and manage user authentication states. Firebase Authentication's GoogleAuthProvider to create a Google provider for sign-in, getAuth to get the authentication instance, and signInWithPopup to initiate the Google sign-in process.
- import app from "../firebase": it represents the Firebase application instance where Firebase services are initialized. Throughout, imports the Firebase app instance from a file that where Firebase services are initialized and configured (authentication, database, etc.).
- import useDispatch from "react-redux": this library can manages application state and dispatches actions to the Redux store.

- import useNavigate from ”react-router-dom”: With this library, can enables navigation and routing within a React application. useNavigate hook to programmatically navigate to different routes within the application.

Code Purpose: The code represents an OAuth component that handles Google sign-in functionality. When user click the ”Continue with Google” button, initiates the Google sign-in process when the is clicked. Server interaction sends user information (name, email, photo) to a backend server via a POST request to authenticate and possibly create a user in the backend system. Upon successful authentication and receiving user data from the backend, dispatches the signInSuccess action to update the Redux store with the user's information. Finally, redirects the user to the home route after a successful sign-in, using the navigate function from React Router DOM.

N. src/Components/PrivateRoute.jsx

Libraries Used and Their Purpose:

- import useSelector from ”react-redux”: it manages application state and provides a way to extract data from the Redux store.
- import Outlet, Navigate from ”react-router-dom”:this handles navigation and routing within the React application.

Code Purpose:

- Uses useSelector to access the currentUser information from the Redux store. This information likely signifies whether a user is authenticated/logged in.
- Route Protection: Acts as a gatekeeper for specific routes/pages, ensuring that only authenticated users can access certain parts of the application. If the user is not logged in, they will be redirected to the sign-in page to authenticate before accessing the protected content.

IX. Directory organization - Backend

The backend is responsible for managing servers and databases. It stores and manages data, as well as handles actions taken by users on the client side of the application. The backend's roles include storing data generated as a result of user behavior in the frontend database (Front Desk) and retrieving data required by users from the frontend database (Front Desk). Location of source code is here.

Directory	File Name	Library
api	controllers models routes utils index.js	dotenv bcryptjs jsonwebtoken mongoose express
api/controllers	ai.controllers.js auth.controllers.js listing.controllers.js user.controllers.js	
api/models	ai.models.js listing.models.js user.models.js	
api/routes	ai.route.js auth.route.js listing.route.js mortgage.route.js user.route.js	react
api/routes	error.js verifyUser.js	

Fig. 7: Directory Organization - Backend

A. controllers/auth.controller.js

Libraries Used and Their Purpose:

- User from ”../models/user.model.js”: This import brings in the User model from the user.model.js file, enabling interactions with the user database schema and data.
- bcryptjs: Imported for password hashing. It provides functions for hashing and comparing passwords securely, enhancing user password security in storage and authentication.
- errorHandler from ”../utils/error.js”: Importing an error handler utility function. This utility might encapsulate error generation and formatting for consistent error responses across the application.
- jsonwebtoken (jwt): Used to create and manage JSON Web Tokens (JWTs). This library assists in generating tokens upon user authentication.

Code Purpose: These controllers handle user authentication, both through traditional signup/signin and third-party authentication via Google, ensuring secure user management and interaction with the application's authentication system.

B. controllers/listing.controller.js

Libraries Used and Their Purpose:

- Listing from ”../models/listing.model.js”: Imports the Listing model from the listing.model.js file, allowing interaction with the listing database schema and data.
- errorHandler from ”../utils/error.js”: Imports an error handler utility function, potentially encapsulating error generation and formatting for consistent error responses across the application.
- mongoose: Utilized for MongoDB object modeling in the Node.js environment. It provides a straightforward schema-based solution to model application data and interact with the MongoDB database.

Code Purpose: Each controller serves a distinct purpose, from creating and updating listings to fetching individual or multiple listings based on specified criteria, ensuring the effective management and retrieval of listing data within the application.

C. controllers/mortgage.controller.js

Libraries Used and Their Purpose:

- errorHandler from "../utils/error.js": Imports an error handler utility function, potentially for consistent error handling and formatting across the application.
- onnxruntime-node: This library provides functionality for loading and executing ONNX models.

Code Purpose:

- The main purpose of this code snippet is to handle mortgage-related functionalities by utilizing an AI model
- Defines a function that loads the AI model using onnxruntime-node. It specifies the path to the ONNX model file
- postFeatures handles a POST request containing features related to a mortgage. It expects the features to be sent in the request body.

D. controllers/user.controller.js

Libraries Used and Their Purpose:

- bcryptjs: Imported for password hashing, ensuring sensitive user information like passwords are securely stored.
- User from "../models/user.model.js": Imports the user model, allowing interaction with the database regarding user information.

Code Purpose: This function incorporates error handling to manage potential issues like unauthorized access, missing users, or database-related errors.

E. models/listing.model.js

Libraries Used and Their Purpose: mongoose: Used to define the structure and schema for MongoDB documents. It allows seamless interaction with MongoDB and simplifies the creation of models and schemas

Code Purpose: The schema is then used to create a Mongoose model named "Listing," which corresponds to this defined schema. This model allows for interactions with the MongoDB database, including creating, updating, deleting, and querying listing documents.

F. models/user.model.js

Code Purpose: This code represents the structure of the user data and provides methods to interact with the MongoDB database, including CRUD (Create, Read, Update, Delete) operations on user documents. Finally, the User model is exported for use in other parts of the application.

G. routes/auth.routes.js

Libraries Used and Their Purpose: express: used to build web applications and APIs by providing a set of robust features for web development.

Code Purpose: Purpose of this code is configured to handle specific authentication actions like user registration, login, Google authentication, and sign-out by mapping them to their respective controller functions defined in auth.controller.js. The router is then exported for use in the application.

H. routes/listing.routes.js

Code Purpose: Purpose of this code is to configured to handle various operations related to listings and are protected by the verifyToken middleware to ensure authenticated access to specific endpoints. The router is then exported for use in the application.

I. routes/mortgage.routes.js

Libraries Used and Their Purpose: express: used for building web applications and APIs by providing a set of robust features for web development.

Code Purpose: Configures a POST route Mortgage calculator that corresponds to the postFeatures function in the mortgage.controller.js file. This route handles the submission of features required for mortgage calculation.

J. routes/user.routes.js

Code Purpose: Purpose of this code is to defines routes for operations like updating user details, deleting users, fetching user information, and retrieving user-specific listings, all encapsulated within the user-related endpoints.

K. utils/error.js

Code Purpose: Purpose of this code is to creates and returns an error object with specific status codes and messages. It's a utility that centralizes the creation of custom errors to maintain consistency in error formatting across the application.

L. utils/verifyUser.js

Libraries Used and Their Purpose:

- jsonwebtoken: Imported to verify and decode JSON Web Tokens (JWTs) received from the request cookies, allowing authentication checks based on the token's validity and extracting user information from it.
- errorHandler from "./error.js": Utilizes a custom error handler imported from a local file. It's invoked to generate specific error objects with defined status codes and messages for various authentication scenarios.

Code Purpose: This function serves as middleware for protected routes, validating incoming JWTs from request cookies. It checks token existence, verifies its validity using the jsonwebtoken library, and sets req.user with decoded user information if authentication is successful. If the token is missing or invalid, it triggers appropriate error handling using the custom errorHandler.

M. index.js

Libraries Used and Their Purpose:

- dotenv: Manages environment variables for configuration settings.
- cookie-parser: Parses cookies attached to the client's requests for easy accessibility.

Code Purpose:

- Server Setup: Establishes server connectivity, handling connections to MongoDB while hiding sensitive information such as passwords.

- Routing Configuration: Defines routes for various functionalities (users, authentication, listings) within the API using the routers from separate files.
- Error Handling Middleware: Captures errors across the application, ensuring consistent error responses are sent back to clients with appropriate status codes and messages.

X. Use Cases

A. Searching new home

1. When a user visits the HomeFinder website, they are presented with an inviting "Get started today!" button on the homepage.

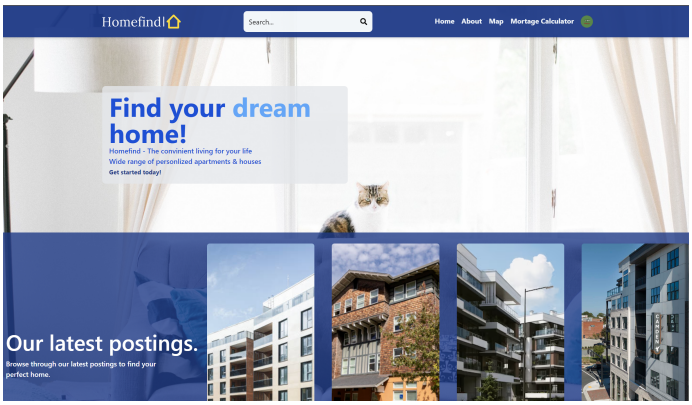

Fig. 8: Welcomepage

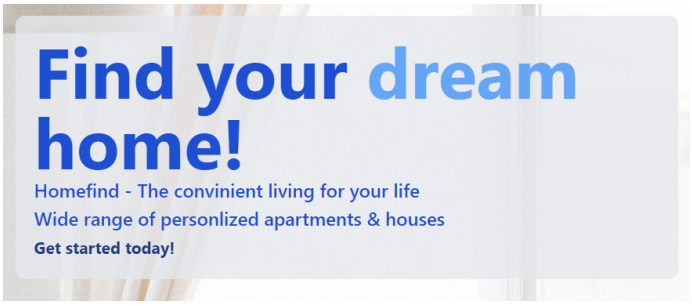

Fig. 9: "Get started today"

2. Clicking this button directs them to a search interface abundant with filtering options, allowing for a highly customized property search experience.

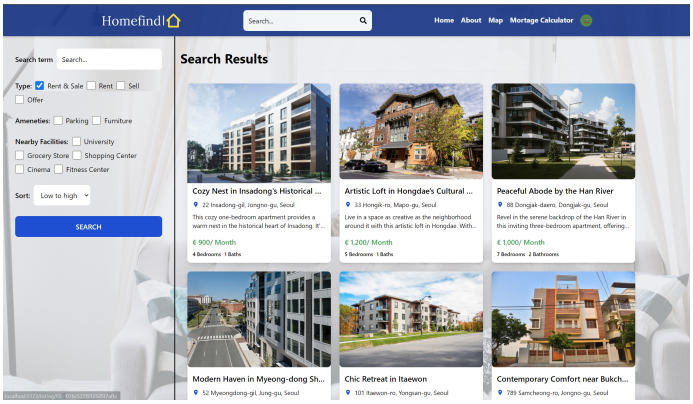

Fig. 10: Searchpage

3. The user can apply a range of filters on HomeFinder to narrow down their search according to their specific needs. After selecting the desired criteria, they execute the search by clicking the "search" button, prompting HomeFinder's system to display a list of properties that match the selected parameters.

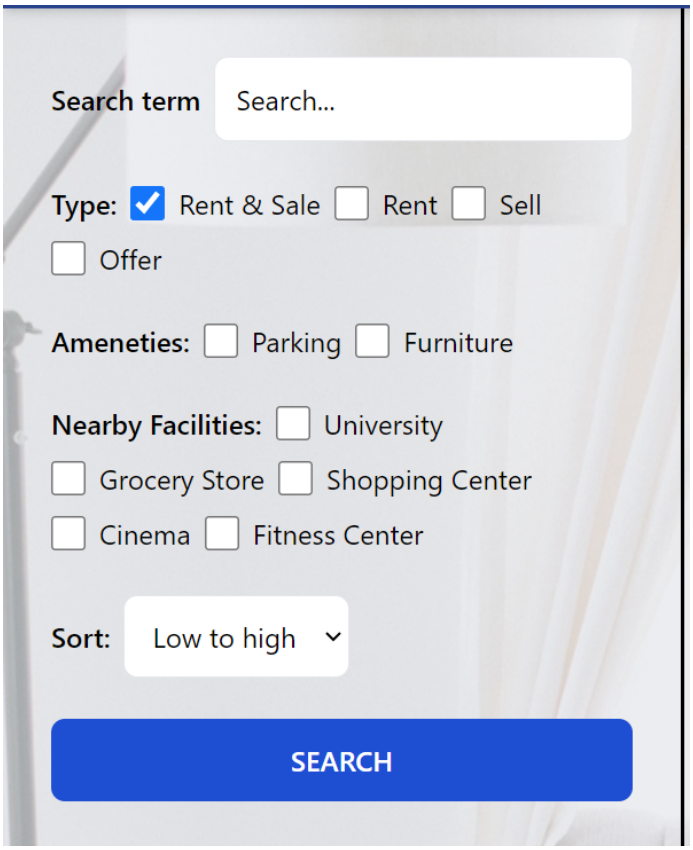

Fig. 11: Filters

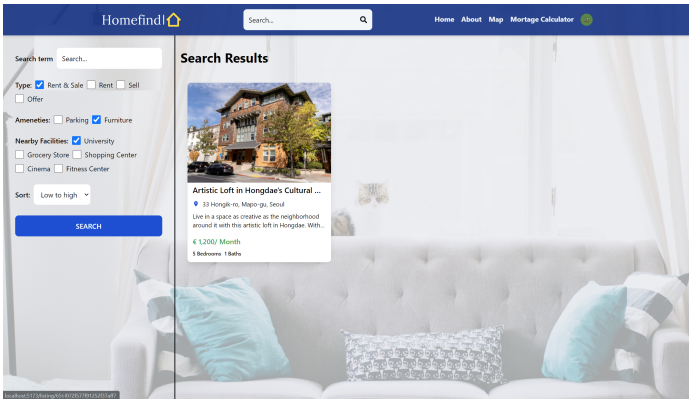

Fig. 12: Search Results

4. If a particular property catches the user's interest, they can click on it to learn more. Alternatively, if the results do not meet their expectations, they have the option to modify the filters and perform another search. HomeFinder offers this iterative search process to help users find their ideal property.

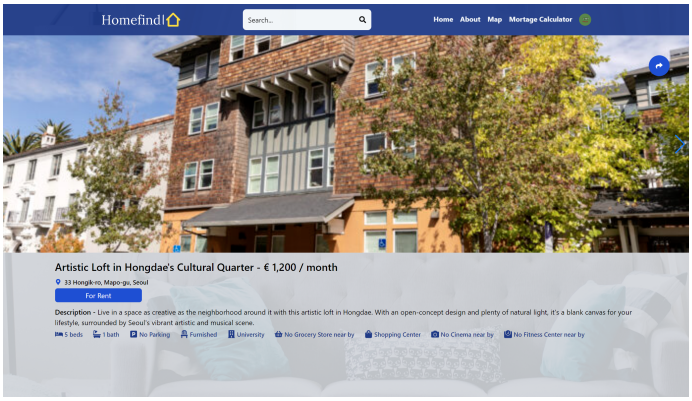

Fig. 13: Apartment description

5. Once the user finds a property they are interested in on HomeFinder, they can contact the seller directly through the platform, facilitating a straightforward communication channel for further inquiries or to initiate the buying process. HomeFinder serves as a comprehensive tool, guiding users from the beginning of their property search journey to the point of contact with sellers.

XI. Software Installation Guide

XII. Discussion